Escola Politècnica Superior
de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL DE FI DE CARRERA

TÍTOL DEL TFC : Desarrollo de un sistema de transmisión de datos

TITULACIÓ: Ingeniería Técnica Aeronáutica, especialidad Aeronavegación

AUTOR: Daniel de Miguel Serrano

DIRECTOR: Dagoberto José Salazar

DATA: June 25, 2007

Títol : Desarrollo de un sistema de transmisión de datos

Autor: Daniel de Miguel Serrano

Director: Dagoberto José Salazar

Data: 25 de junio de 2007

Resum

Este TFC o Trabajo Final de Carrera se basa en el desarrollo de una aplicación para lograr un sistema de transmisión de datos que será utilizado en la asignatura de NACC (Navegación Aérea, Cartografía y Cosmografía) en la EPSC (Escuela Politécnica Superior de Castelldefels).

El objetivo principal de este proyecto es diseñar y construir una herramienta de software para establecer la comunicación inalámbrica entre dos dispositivos diferentes; un ordenador portátil con un receptor GPS o bien con una tarjeta de control de servomotores. Esta herramienta de software utiliza los módulos MHX 920 para establecer el sistema de comunicación de datos entre los dispositivos conectados a las tarjetas.

Además, dos experimentos diferentes fueron realizados. Estos experimentos proponen la creación de dos estaciones diferentes, una fija que es la estación base y otra móvil con un dispositivo a bordo. El primer experimento, que incorpora un receptor GPS como dispositivo, clasifica la información provinente del GPS recibida como sentencias NMEA, muestra la latitud y la longitud del receptor y muestra la posición de receptor GPS en un mapa. El segundo, usa una tarjeta de control de servomotores para conducir un coche de control remoto utilizando este sistema de transmisión de datos.

Title : Data transmission system development

Author: Daniel de Miguel Serrano

Director: Dagoberto José Salazar

Date: June 25, 2007

Overview

This Final Project (TFC-Trabajo de Fin de Carrera) consists in developing an application to achieve a data transmission system that will be used at NACC course (Navegación Aérea, Cartografía y Cosmografía) at the EPSC (Escuela Politécnica Superior de Castelldefels).
The main objective of this project is to design and build a software tool to manage wireless communication between two different devices, a laptop with a GPS receiver or a servo-motor controller board. This software tool has been designed to manage the MHX 920 Spread Spectrum modules in order to establish the wireless data communication system between the devices connected to them.
Furthermore, two different experiments were carried out. These experiments set up two different stations, a fixed one which is the base station and a mobile one with a device on board. The first experiment, which included a GPS receiver as device, classifies GPS information received as NMEA sentences, shows receiver latitude and longitude and plots GPS receiver position into a map. The second one, uses a servo controller board to drive a remote-controlled car by using this data transmission system.

To a real tutor.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

The purpose of this project is to develop an application to achieve a data transmission system that will be used at NACC course (Navegación Aérea, Cartografía y Cosmografía) at the EPSC (Escuela Politécnica Superior de Castelldefels).

The primary objective of this project is to design and construct a software tool to manage wireless communication between two different devices, for instance a GPS receiver and a laptop. Joining MHX 920 Spread spectrum modules to each device, two stations are formed: A mobile one and a fixed one.

The software application of this project has been developed as far as the data to be received from GPS is processed and mobile station position is showed and plotted into a map.

Furthermore, the software has been implemented to control the mobile station movement. Using this wireless data transmission system and a servo controller board, a remote-controlled car movement has been driven by a user at the base station.

In order to test the achievement of the primary objective, two secondary objectives were set, in the form of two practical experiments. Those experiments were:

1. The mobile station, composed by a GPS receiver and a MHX 920 Spread Spectrum module, is moved around Campus de Castelldefels. It sends position information to the base station where it is processed, classified and plotted into a map.
2. The mobile station, composed by a Servo controller board and a MHX 920 module, is remote-controlled by applying the designed software. The mobile station movements are sent by the base station.

The project internal structure is:

- The theoretical basis necessary to manage and integrate the different elements into the project.
- An introduction to hardware elements explaining their characteristics and operating method.
- A manual that explains how to use and implement the software tool.
- Explanation of the experiments carried out.
- Annexes with supplementary information that includes the mathematical formula applied.

CHAPTER 1. THEORY

1.1. Introduction

This section presents the theorical base this project is build on.

1.2. NAVSTAR GPS

1.2.1. Introduction to GPS

The NAVSTAR-GPS[1], better known as GPS, is a GNSS[2] that establish a position at any point of the globe by determining the following two values:

- One's exact location (longitude, latitude and height coordinates) accurate to within a range of 20 m to approximately 1mm.
- The Universal Time Coordinated (UTC), accurate to within a range of 60ns to approx 5ns.

From these values, speed and direction of travel (course) can be derived as well as time. The coordinates and time values are determined by 28 satellites orbiting the Earth.

The satellites are following an IOC[3] with an inclination of 55Â° with respect to the equator. They orbit the Earth every 11 hours and 58 minutes at a height of 20,180 km on 6 different orbital planes.

1.2.2. Determining the position

Each satellite on the constellation transmits its exact location and an extremely accurate time to receivers on earth. With this information, GPS receivers can calculate the distance from the satellite, and by combining this information from four different satellites, the receiver can calculate its exact position by a process knows as trilateration[4].

When you know the distance to a given satellite, and the exact position of that satellite, you know that you are located on a sphere centered on the satellite with a radius equal to your distance to it. With similar information from a second satellite, the position is narrowed to the area held in common by the two spheres.

[1] *NAVSTAR-GPS*: NAVigation System with Timing And Ranging Global Positioning System.
[2] *GNSS*: Global Navigation Satellite System
[3] *IOC*: Intermediate Circular Orbit
[4] *Trilateration* is a method to determine the relative positions of objects using the geometry of triangles and the length of their sides, in a similar fashion as triangulation but using distances instead of angles.

Adding information from a third satellite further narrows down the position to the two points were the three spheres intersect. To determine which point represents the exact receiver location it can make a fourth measurement, but usually one of the two points obtained from three satellites represents an absurd location, such as far out in space, or a point that is moving at an impossible velocity, and can be rejected without measuring. See Figure 1.1 to see three satellite position. The fourth measurement, however, is still needed as explained below.

The distance to the satellites is determined by measuring the amount of time that it takes the radio signal from the satellite to reach the receiver[5]. Because radio signals travel at the speed of light[6] the travel times are extremely short. Therefore you need an extremely precise timing device to measure it accurately, hence the need for atomic clocks on the satellites.

However, the receivers don't have atomic clocks so there will be measurement errors on that side of the system, and even a small timing error can result in a large position error. Here is where the fourth measurement comes in.

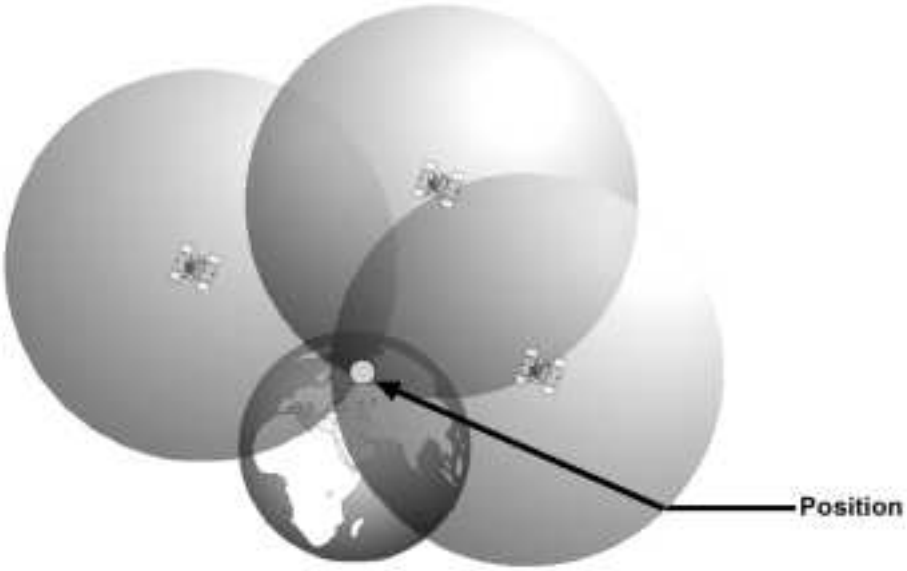

Figure 1.1: Position determined at the point where al three spheres intersect

If all measurements are accurate, the sphere defined by the fourth measurement should intersect the other three spheres at a point representing our true location. If there are errors, however, the fourth sphere will not intersect the other spheres. Because the receiver error will be the same for all four measurements, a computer in the receiver can calculate the correction needed to make all four spheres intersect, and then apply that correction to the measurements to determine the true position.

[5] Distance = velocity X time

[6] 299792.458 km per second

1.2.2.1. Other errors

Different errors enter into the GPS signal measurements. These include:

- Atmospheric interference.
- Reflection from ground obstacles.

Several techniques such as dual frequency correction, signal filters, and mathematical modeling are used to minimize these errors.

1.2.2.2. Increased accuracy

Techniques are constantly being developed to make GPS systems even more accurate and reliable. The most widely used at this time are differential GPS (DGPS) and SBAS[7]; WAAS and EGNOS are examples from North America and Europe respectively.

DGPS basically uses land based stations, whose locations are exactly known, to receive GPS signals, apply corrections (since they know exactly were they are, they can determine the errors from the GPS signals) and broadcast them so that properly equipped receivers can use them to correct their own signals.

WAAS[8] is a secondary satellite system developed by the Federal Aviation Administration that not only transmits GPS signals, but also monitors and reports the status of GPS satellites, and transmits differential GPS information. The system is available only over North America and parts of the Pacific Ocean.

EGNOS[9] is a SBAS under development by the European Space Agency, the European Commission and EUROCONTROL. It is intended to supplement the GPS, GLONASS and Galileo (when it becomes operational) systems by reporting on the reliability and accuracy of the signals. According to specifications, horizontal position accuracy should be better than 7 metres.

Most GPS units available today are capable of accuracies of about 20 meters or less (See Table 1.1). Use of advanced techniques such as the ones mentioned above can bring accuracies down to a meter or less. Specialized applications which employ sophisticated data manipulation techniques and high end equipment can achieve accuracies measured in centimeters.

Horizontal Precision	**Vertical precision**	**Time Precision**
$\leq$ 13 m	$\leq$ 22 m	$\approx$ 40 ns

Table 1.1: Typical GPS Precision

[7] *SBAS*: Satellite Based Augmentation System.
[8] *WAAS*: Wide Area Augmentation System.
[9] *EGNOS*: European Geostationary Navigation Overlay Service.

1.2.3. Data format: NMEA

In order to relay computed GPS variables such as position, velocity, course, etc. to a peripheral, eg: computer, screen, transceiver, etc., GPS modules usually have a serial interface[10]. Then, the most important elements of receiver information are broadcast via this interface in a special data format. This format is standardised by the National Marine Electronics Association (NMEA) to ensure that data exchange takes place without any problems.

This type of data includes the complete PVT (position, velocity, time) solution computed by the GPS receiver. The idea of NMEA is to send a line of data called a *sentence*, that is totally self-contained and independent from other sentences. The data itself is ASCII text.

Each sentence begins with a '$' and ends with a carriage return/line feed sequence, that can't be longer than 80 characters of visible text (plus the line terminators). The data is contained within this single line with data items separated by commas.

After the '$' symbol, all of the standard sentences have a two-letter prefix that defines the device that uses that sentence type. In the case of GPS receivers, the prefix is 'GP'. It is followed by a three-letter sequence that defines the sentence content.

There is a provision for a checksum at the end of each sentence which may or may not be checked by the unit that reads the data. The checksum field consists of a '*' and two hex[11] digits representing an 8 bit exclusive OR of all characters between, but not including, the '$' and '*'. The checksum allows the user to check the truthfulness of the information received. It is required on some sentences.

Table 1.2 presents a description of NMEA data blocks.

Field	**Description**
$	Start of the data set
GP	Information originating from a GPS appliance
DTS	Data set identifier
inf1 bis inf n	Information with number 1 to ... n
,	Comma used as a separator for different items of information
*	Asterisk used as a separator for the checksum
CS	Checksum (control word) for checking the entire data set
<CR> <LF>	End of the data set: carriage return (<CR>) and line feed, (<LF>)

Table 1.2: Description of the individual NMEA Data Set Blocks

The most common GPS NMEA sentences are listed below. The ones used in this project are emphasised by:

- Using italic font for them if used by the GPS receiver.

[10] Serial communication TTL or RS232.

[11] *Hex*: Hexadecimal, base-16, or simply hex, is a numeral system with a radix, or base, of 16, usually written using the symbols 0 to 9 and A to F, or a to f.

- Underlining them if defined but not used.

1. AAM - Waypoint Arrival Alarm
2. ALM - Almanac data
3. APA - Auto Pilot A sentence
4. APB - Auto Pilot B sentence
5. BOD - Bearing Origin to Destination
6. BWC - Bearing using Great Circle route
7. DTM - Datum being used.
8. *GGA - GPS Fixed Data*
9. *GLL - Geographic Position: Latitude/Longitude*
10. GRS - GPS Range Residuals
11. *GSA - Overall Satellite data*
12. GST - GPS Pseudorange Noise Statistics
13. *GSV - GNSS Satellites in view, detailed satellite data*
14. MSK - send control for a beacon receiver
15. MSS - Beacon receiver status information.
16. RMA - recommended Loran data
17. RMB - recommended navigation data for GPS
18. *RMC - recommended minimum specific data for GPS*
19. RTE - route message
20. TRF - Transit Fix Data
21. STN - Multiple Data ID
22. VBW - dual Ground / Water Spped
23. *VTG - Course over Ground and Ground Speed, horizontal course and velocity)*
24. WCV - Waypoint closure velocity (Velocity Made Good)
25. WPL - Waypoint Location information
26. XTC - cross track error
27. XTE - measured cross track error

28. ZTG - Zulu (UTC) time and time to go to destination
29. *ZDA - Date and Time*

In section A.2., tables A.1 to A.10, the reader may find additional description of some of these NMEA sentence types.

1.3. Telemetry

1.3.1. Introduction to Telemetry

Telemetry is a technology that allows the remote measurement and reporting of information of interest to the system designer or operator. The word is derived from the Greek words *tele* = remote, and *metron* = measure. Systems that need instructions and data sent to them in order to operate require the counterpart of telemetry, telecommand.

Telemetry typically refers to wireless communications (i.e. using a radio frequency system to implement the data link), but can also refer to data transfer over other media, such as a telephone or computer network or via an optical link.

1.3.2. Wireless communication

The term wireless is normally used to refer to any type of electrical or electronic operation which is accomplished without the use of a ”hard wired” connection, though they may be accomplished with the use of wires if desired.

Wireless communication is the transfer of information over a distance without the use of electrical conductors or “wires”. The distances involved may be short (a few meters as in television remote control) or very long (thousands or even millions of kilometres for radio communications). When the context is clear the term is often simply shortened to ”wireless”. Wireless communications is generally considered to be a branch of telecommunications.

The term wireless technology is generally used for mobile IT equipment. It encompasses cellular telephones, personal digital assistants (PDAs), and wireless networking. Other examples of wireless technology include GPS units, garage door openers and or garage doors, wireless computer mice and keyboards, satellite television and cordless telephones.

Wireless communication is developed in this project by using the MHX 920 modules.

1.4. Datum

If you are comparing GPS coordinates to a chart or map, the map datum in the GPS unit must be set to match the chart or map's datum for accurate comparison. The datum used to create a chart or map is generally listed in the legend.

1.4.1. Introduction to Datum

A datum[12] is a reference from which measurements are made. In geodesy, a datum is a mathematical model of the Earth which approximates the shape of the Earth, and enables calculations such as position and area to be carried out in a consistent and accurate manner. The datum is physically represented by a framework of ground monuments, whose positions have been accurately measured and calculated on this reference surface.

A reference datum is a known and constant surface which can be used to describe the location of unknown points on the Earth. Since reference datums can have different radii and can have different centre points, a specific point on Earth can have substantially different coordinates depending on the datum used to make the measurement. On Earth the normal reference Datum is mean sea level.

The NGS[13] defines geodetic datum as:

1. "A set of constants specifying the coordinate system used for geodetic control, i.e., for calculating the coordinates of points on the Earth."
2. "The datum, as defined before, together with the coordinate system and the set of all points and lines whose coordinates, lengths, and directions have been determined by measurement or calculation."

These differing definitions require caution when using the word "datum." The first definition makes datum synonymous with the selection of a reference coordinate system (origin and orientation). The second definition makes datum synonymous with a list of coordinates of the control points. When the first definition is used, the published coordinates of control points can change when better measurements allow better determinations. With the second definition, a change in coordinates should result in a new datum. The most common reference Datums in use are NAD83, ED50, and WGS84.

Ellipsoidal earth models are required for accurate range and bearing calculations over long distances. Loran-C, and GPS navigation receivers use ellipsoidal earth models to compute position and waypoint information. Ellipsoidal models define an ellipsoid with an equatorial radius and a polar radius. The best of these models can represent the shape of the earth over the smoothed, averaged sea-level surface to within about one-hundred meters.

[12] Plural Datums or Data.
[13] *NGS*: National Geodetic Survey.

1.4.2. WGS-84

WGS-84[14] is the reference datum used by the U.S. DoD[15] and is defined by the National Geospatial-Intelligence Agency[16]. WGS 84 is used by DoD for all its mapping, charting, surveying, and navigation needs, including its GPS “broadcast” and “precise” orbits.

Older american datums[17] no longer provided sufficient data, information, geographic coverage, or product accuracy for all the then current and anticipated applications. The means for producing a new WGS were available in the form of improved data, increased data coverage, new data types and improved techniques. GRS 80 parameters together with available Doppler, satellite laser ranging and VLBI (Very Long Baseline Interferometry) observations constituted significant new information.

WGS-84 was defined in January 1987 using Doppler satellite surveying techniques. It was used as the reference frame for broadcast GPS Ephemerides (orbits) beginning January 23, 1987.

At 0000 GMT[18] January 2, 1994, WGS-84 was upgraded in accuracy using GPS measurements. The formal name then became WGS-84 (G730)[19]. It became the reference frame for broadcast orbits on June 28, 1994. At 0000 GMT September 30, 1996 (the start of GPS Week 873), WGS 84 was redefined again and was more closely aligned with IERS[20], ITRF[21]. It is now formally called WGS-84 (G873). WGS-84 (G873) was adopted as the reference frame for broadcast orbits on January 29, 1997.

The WGS-84 datum originally used the GRS 80 reference ellipsoid, but has undergone some minor refinements in later editions since its initial publication. Most of these refinements are important for high-precision orbital calculations for satellites, but have little practical effect on typical topographical uses. Table 1.3 lists the primary ellipsoid parameters.

1.4.3. ED50

ED50[22] is a geodetic datum which was defined after World War II for the international connection of geodetic networks.

Some of the important battles of World War II were fought on the borders of Spain, the Netherlands, Belgium and France, and the mapping of these countries had incompatible latitude and longitude positioning. This led to the setting up of ED50 as a consistent mapping datum for much of Western Europe. It was, and still is, used in much of Western

[14] *WGS84*: World Geodetic System of 1984.
[15] *U.S. DoD*: U.S. Department of Defense.
[16] *NGA*: Formerly the National Imagery and Mapping Agency or Defense Mapping Agency.
[17] WGS-66, WGS-72.
[18] *GMT*: Greenwich Mean Time.
[19] Since the upgrade date coincided with the start of GPS Week 730.
[20] *IERS*: International Earth Rotation Service.
[21] *ITRF*: International Terrestrial Reference Frame.
[22] *ED50*: European Datum 1950.

Europe apart from Great Britain, Ireland, Sweden and Switzerland, which have their own datums.

It was based on the International Ellipsoid of 1924[23] (radius of the Earth's equator 6378.388 km), and widely used all over the world up to the 1980s, when GRS80 and WGS84 were established.

Many national coordinate systems of Gauss-Krüger are defined by ED50 and oriented by means of Geodetic Astronomy. Up to now it has been used in data bases of gravity field, cadastre, small surveying networks in Europe and America, and by some developing countries with no modern baselines.

The geodetic datum of ED50 is centred at the Frauenkirche of Munich in Southern Germany, where the approximate centre of the Western Europe national networks was situated in the years of the cold war. ED50 was also part of the fundamentals of the NATO coordinates (Gauss-Krüger and UTM) up to the 1980s.

1.4.4. From WGS84 to ED50

The longitude and latitude lines on the two datums are the same in the Archangel region of north-west Russia. As one moves westwards across Europe, the longitude lines on ED50 gradually become further west than their WGS84 equivalents, and are around 100 metres west in Spain and Portugal. Moving southwards, the latitude lines on ED50 gradually become further south than the WGS84 lines, and are around 100 m south in the Mediterranean Sea[24].

To pass from datum WGS84 to ED50, the differences in Cartesian coordinates considered in transformation are given by Table 1.3.

Datum	**x**	**y**	**z**	**Semi-major axis (a)**	**Eccentricity (e^2)**
WGS84	0	0	0	6378137.0 m	0,00669437999014
ED50	-84 m	-107 m	-120 m	6378388.0 m	0,00672267002233

Table 1.3: Datum properties

1.5. Programming

1.5.1. Introduction to Programming

Computer programming[25] is the process of writing, testing, and maintaining the source code of computer programs. The source code is written in a programming language. This

[23] "Hayford-Ellipsoid" of 1909.

[24] If the lines are further west, the longitude value of any given point becomes more easterly. Similarly, if the lines are south, the values become northerly.

[25] Often shortened to programming or coding.

code may be a modification of existing source or something completely new, the purpose being to create a program that exhibits the desired behaviour.

Within software engineering, programming is regarded as one phase in a software development process. As an introduction to the project's code created we will explain some necessary programming characteristics:

1. Programming languages. → C++
2. Programming paradigm. → OOP
3. Fundamental concepts.

1.5.2. Programming languages

A programming language is an artificial language that can be used to control the behaviour of a machine, particularly a computer. Programming languages, like human languages, are defined through the use of syntactic and semantic rules, in order to determine structure and meaning respectively.

Programming languages are used to facilitate communication about the task of organising and manipulating information, and to express algorithms precisely. Some authors restrict the term ”programming language” to those languages that can express all possible algorithms; sometimes the term ”computer language” is used for more limited artificial languages.

Thousands of different programming languages have been created, and new languages are created every year. The programming language used in this project is C++.

1.5.2.1. C++ Programming

C++[26] is a general-purpose, high-level programming language with low-level facilities. It is a statically typed free-form multi-paradigm language, supporting procedural programming, data abstraction, object-oriented programming, and generic programming.

Since the 1990s, C++ has been one of the most popular commercial programming languages.

1.5.3. Programming paradigm

A programming paradigm is a fundamental style of programming regarding how problems solutions are to be formulated in a programming language.

[26] Called “c plus plus” and originally named “C with classes”. Developed by Bjarne Stroustrup at Bell Labs.

A programming paradigm provides and determines the view that the programmer has of the execution of the program.

1.5.3.1. Object Oriented Programming

Object-oriented programming (OOP) is a programming paradigm that uses objects[27] to design applications and computer programs. Nowadays, many popular programming languages support OOP.

These objects act on each other, as opposed to a traditional view in which a program may be seen as a collection of functions, or simply as a list of instructions to the computer.

In OOP, objects are capable of receiving messages, processing data, and sending messages to other objects. Each object can be viewed as an independent little machine or actor with a distinct role or responsibility.

Object-oriented programming may be seen as building a collection of cooperating objects, as opposed to the traditional view in which a program may be seen as a list of instructions to the computer.

Object-Oriented Programming is intended to promote greater flexibility and maintainability in programming, and is widely popular in large-scale software engineering. By virtue of its strong emphasis on modularity, object oriented code is intended to be simpler to develop and easier to understand later on, lending itself to more direct analysis, coding, and understanding of complex situations and procedures than less modular programming methods.

1.5.3.2. Fundamental concepts

In programming, and specifically in OOP, there are different concepts that should be explained. Main concepts used in this project are:

Class A class defines the abstract characteristics of an object, including:

- attributes
- fields
- properties

and the functions it can perform:

- behaviours
- methods
- features.

[27] Individual unit of run-time data storage that is used as the basic building block of programs.

Object A particular instance of a class. The class defines all possible characteristics of objects by listing the characteristics that they can have; the object is one particular class, with particular versions of the characteristics. The set of values of the attributes of a particular object is called its state.

Method An object's ability. It is defined as a class characteristic but it should only affect one particular object. It is a specific function of an object.

Message passing The process by which an object sends data to another object or asks the other object to invoke a method. Also known to some programming languages as interfacing.

Inheritance Allows the sub-class creation, which is a more specialised version of a class. From a defined class, you can create others that have the characteristics and functions of the primitive one and allows the definition of new ones.

Encapsulation Encapsulation conceals the exact details of how a particular class works from objects that use its code or send messages to it. Encapsulation is achieved by specifying which classes may use the accessible members of an object. Class members -functions or characteristics- can be declared as:

public Available to all classes.

protected Available to sub-classes and the defining class.

private Available only to the defining class.

Polymorphism Allows a single definition to be used with different types of data (specifically, different classes of objects). For instance, a polymorphic function definition can replace several type-specific ones, and a single polymorphic operator can act in expressions of various types. Typical operators that can be used in polymorphism are +,-,*, among others.

1.6. Serial Port

1.6.1. Introduction to PC connectivity

Connectivity is the property of a device that is enabled to be connected, generally to a PC or another device.

A computer connector is any hardware device used to connect computers to networks, printers or other devices. Generally, these connectors have specific names that provide more precise identification, and use of those names is encouraged. Most computer connectors are electrical or optical.

All external devices connect to the computer's system unit via cables and ports[28]. Typical PC ports are presented in Table 1.4.

[28] Slot into which you plug a cable.

PS/2	AT keyboard
Serial	Parallel
VGA	Game
USB	Speakers
Microphone	Line in
Ethernet	

Table 1.4: PC Ports list

The ANTAARIS GPS receiver, the MHX 920 modules and the servo controller board, all have a serial port connector, therefore information is related to this port type.

1.6.2. Serial Port communication

Serial communications is the process of sending data one bit at a time, sequentially, over a communications channel or computer bus.

A serial port sends and receives data one bit at a time over one wire. While it takes eight times as long to transfer each byte of data this way, only a few wires are required. In fact, two-way (full duplex) communications is possible with only three separate wires - one to send, one to receive, and a common signal ground wire.

The serial port on a PC is a full-duplex device, meaning that it can send and receive data at the same time. In order to be able to do this, it uses separate lines for transmitting and receiving data. Some types of serial devices support only one-way communications and therefore use only two wires in the cable - the transmit line and the signal ground.

Serial communication requires that you specify the following four parameters:

- Baud Rate of the transmission.
- Number of data bits encoding a character.
- Sense of the optional parity bit.
- Number of stop bits.

1.6.2.1. Baud Rate

Baud[29] rate is a measure of how fast data is moving between instruments that use serial communication. RS-232 uses only two voltage states, called MARK and SPACE. In such a two-state coding scheme, the baud rate is identical to the maximum number of bits of information, including ”control” bits, that are transmitted per second. MARK is a negative voltage and SPACE is positive.

Typical baud rates are shown in Table 1.5.

[29] Measured in BPS (bits per second).

300	4800
600	7200
1200	**9600**
1800	14400
2400	19200
3600	28800
	38400

Table 1.5: Serial Baud Rates

1.6.2.2. Number of Data Bits

Data bits are transmitted “upside down and backwards”. That is, inverted logic is used and the order of transmission is from least significant bit (LSB) to most significant bit (MSB). To interpret the data bits in a character frame, you must read from right to left, and read 1 for negative voltage and 0 for positive voltage.

The number of Data Bits can usually be set to:

- 5
- 6
- 7
- **8**

1.6.2.3. Parity Bits

An optional parity bit follows the data bits in the character frame. The parity bit is a binary digit that indicates whether the number of bits with value of one in a given set of bits is even or odd. Parity bits are used as the simplest error detecting code.

There are two types of parity bits:

Even parity bit: Set to 1 if the number of ones in a given set of bits is odd; making the total number of ones even.

Odd parity bit: Set to 1 if the number of ones in a given set of bits is even; making the total number of ones odd.

The parity bit can also be set to **none**.

1.6.2.4. Stop Bits

The last part of a character frame consists of 1, 1.5, or 2 stop bits. These bits are always represented by a negative voltage. If no further characters are transmitted, the line stays

in the negative (MARK) condition. The transmission of the next character frame, if any, is heralded by a start bit of positive (SPACE) voltage.

1.6.3. Common configuration

Finally, as one of the most common default configuration, the serial port configuration used in this project is the following:

- Baud Rate: 9600
- Data Bits: 8
- Parity Bit: N
- Stop Bit: 1

1.6.4. Serial Null modem

Null modem is a communication method to connect two DTEs[30] directly using a RS-232 serial cable.

The original RS-232 standard only defined the connection of DTEs with DCEs[31] In a null modem connection, the transmit and receive lines are crosslinked. Depending on the purpose, sometimes one or more handshake lines are crosslinked. See Figure 1.2.

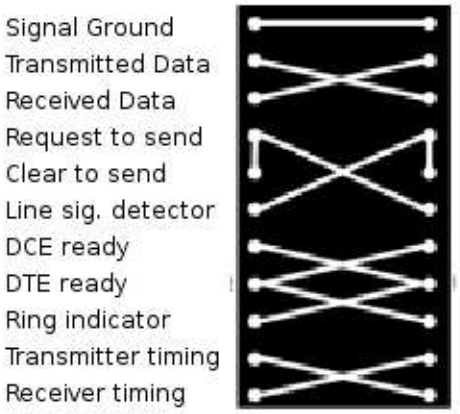

Figure 1.2: Null modem internal connexions

This cable is necessary because both the MHX 920 and the ANTARIS SBEKit are DCEs. To connect them it is required to change their female serial port to a male one, and connect them with the null modem cable. The same occurs with the servo controller board and the MHX 920.

[30] *DTE*: Data Terminal Equipment, such as computers, printers, etc.
[31] *DCE*: Data Circuit-Terminating Equipment, such as modems, MHX 920, etc.

1.7. Servomotor

A servomotor (or servo for short) is a small device that has an output shaft. This shaft can be positioned to specific angular positions by sending the servo a coded signal. As long as the coded signal exists on the input line, the servo will maintain the angular position of the shaft. As the coded signal changes, the angular position of the shaft changes.

In practise, servos are used in radio controlled airplanes or cars to position control surfaces and other mechanisms. They are also used robots.

Servos are extremely useful in robotics. The motors are small, they have built-in control circuitry, and are extremely powerful for their size. It also draws power proportional to the mechanical load. A lightly loaded servo, therefore, does not consume much energy.

Servos are powered and controlled by three wires connection. One is for power[32], another one is for ground and last one is the control wire[33].

1.7.1. How do they work

The servo motor has some control circuits and a potentiometer[34] that is connected to the output shaft.

This potentiometer allows the control circuitry to monitor the current angle of the servo motor. If the shaft is at the correct angle, then the motor shuts off. If the circuit finds that the angle is not correct, it will turn the motor the correct direction until the angle is correct. The output shaft of the servo is capable of travelling somewhere around 180 degrees. Usually, its somewhere in the 210 degree range, but it varies by manufacturer.

A normal servo is used to control an angular motion of between 0 and 180 degrees, and it is not capable of turning any farther due to a mechanical stop built on to the main output gear.

The amount of power applied to the motor is proportional to the distance it needs to travel. So, if the shaft needs to turn a large distance, the motor will run at full speed. If it needs to turn only a small amount, the motor will run at a slower speed. This is called proportional control.

1.7.2. Modifying the angle

The control wire is used to communicate the angle. The angle is determined by the duration of a pulse that is applied to the control wire. This is called Pulse Coded Modulation.

[32] Typically 5 volts.

[33] Wires colours can be different depending on the servo manufacturer. In this case, the configuration is: red for power, black for ground and white for control.

[34] *Potentiometer*: A variable resistor.

The servo expects to see a pulse every 20 milliseconds. The length of the pulse will determine how far the motor turns. A 1.5 millisecond pulse, for example, will make the motor turn to the 90 degree position (often called the neutral position). If the pulse is shorter than 1.5 ms, then the motor will turn the shaft to closer to 0 degress. If the pulse is longer than 1.5ms, the shaft turns closer to 180 degress.

As you can see in Figure 1.3, the duration of the pulse dictates the angle of the output shaft (shown as the circle with the arrow). Note that the times here are standard, and the actual timings depend on the motor manufacturer. The principle, however, is the same.

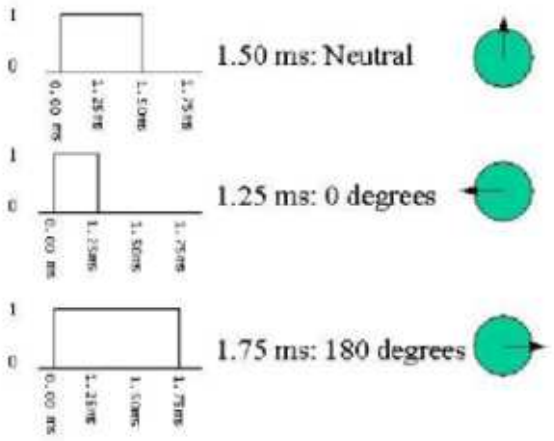

Figure 1.3: Servo's pulse code

CHAPTER 2. HARDWARE

2.1. ANTARIS SBEKit

The ANTARIS SBEKit is the GPS receiver with Dead Reckoning evaluation kit manufactured by ANTARIS, and the one used in this project. This GPS has been used on account of its availability given that the EPSC/UPC owns one of these kits.

The ANTARIS GPS technology supports NMEA-0183. The ANTARIS SBEKit provides NMEA sentences via serial port. Table 2.1 shows ANTARIS default configuration. Please note that Port 1 sends NMEA protocol messages at 9600 Baud Rate.

Parameter	Description	Remark
Port 1, Input	UBX protocol, NMEA and RTCM at 9600 Baud	
Port 1, Output	UBX and NMEA protocol at 9600 Baud	Only NMEA messages activated
Port 2, Input	UBX protocol, NMEA and RTCM at 57600 Baud	
Port 2, Output	UBX and NMEA protocol at 57000 Baud	Only UBX messages activated

Table 2.1: Default configuration

As summary, by default the ANTARIS SBEKit gives GPS information using the NMEA protocol via serial port #1.

2.2. MHX920 module

The MHX920 Spread Spectrum OEM[1] Module is the wireless module used in this project because the EPSC owns two of them. These are the modules wanted to be configured for wireless communication.

The MHX920 operates in the 902-928 MHz ISM[2] band, using frequency-hopping spread-spectrum[3] the module is capable of providing reliable wireless data transfer between almost any equipment which uses an RS232 interface.

While a pair of MHX920 modules can link two terminals devices (*Point-to-Point* operation), multiple modules can be used together to create networks of various topologies, including *Point-to-MultiPoint* and *Repeater* operations. In view of the fact of having only two modules, the operation established has been the first one: *Point-to-Point*.

[1] *OEM*: Original equipment manufacturer: It is a term that refers to a situation in which one company purchases a manufactured product from another company and resells the product as its own.

[2] *ISM*: Industrial, Scientific and Medical (radio bands)

[3] *FHSS*: It is a method for transmitting radio signals by rapidly switching a carrier among many frequency channels, using a pseudorandom sequence known both to transmitter and receiver.

2.2.1. Modes of Operation

The MHX 920 module may be configured to meet a wide range of needs and applications. The module is designed such as all communication is made through one serial port, the RS-232 serial port. It has two operation modes:

Data mode: It provides the asynchronous interface with the host equipment for data sent/received on the RF channel.

Command mode: It is used for configuring and programming the module.

2.2.1.1. Data mode

Data mode is the normal operating mode. When in data mode, the MHX920 communicates with other MHX920 modules. In the *Point-to-Point* configuration there is one **master** module[4] and one **slave** mode[5].

2.2.1.2. Command mode

The MHX920 allows the user to customize the operation of the module through an AT Command Interface; however it is possible to use the AT Commands by manually keying AT Commands and modifying S_Register parameters.

To boot into command mode is very easy. In order to modify the MHX920 configuration, you should follow the next instructions while powering up the MHX920:

1. Connect a straight-through serial cable between DB9 connector and serial port on laptop or PC
2. Run any terminal program such the one remarked here
3. Set the serial port configuration[6]
4. Apply power to the unit
5. While all three RSSI LED's are blinking, type "mhx"[7]
6. Type "AT&V<ENTER>" $\rightarrow$ to know the active configuration.
7. Type the configuration change orders

To return to Data mode, write: "ATA<ENTER>" or "ATO<ENTER>", and to move from one mode to the other write "+++<ENTER>".

[4] The master function is to provide synchronization for the entire network and to control the flow of data.
[5] The slave searches for synchronization with a master.
[6] See section 1.6.3. on page 17.
[7] MHX920 should respond with "OK".

2.2.2. Configuration: AT Commands

The MHX920 has various operating parameters able to be modified using AT Commands. These commands affect the operation of the module in command mode and the transition between data and command modes.[8]

Introduce AT Commands by writing their name and keying <ENTER>. The default settings are emphasized by bold characters. Some important AT Commands are presented in Table 2.2.

Order	Name
ATA	Answer: Puts the module into data mode
ATDxxxxx	Dial changes address to xxxxx
ATH	Dial Report “OK”
ATI	Identification
ATO	On-line Mode into Data mode
AT&F	Load Default Configuration
	&F1 Master PtM (Point-to-Multipoint)
	&F2 Slave PtM
	&F3 Repeater
	&F6 Master PtP (Point-to-Point)
	&F7 Slave PtP
AT&D	Data Terminal Ready
	&D0 DTR line ignored
	&D1 Not supported
	&D1 Force into command mode
AT&K	Handshaking.
	&K0 Disable handshaking
	&K2 Not supported
	&K3 Enable hardware handshaking
AT&S	Data Set Ready
	&S0 DSR always on
	&S1 DSR on in data mode
	&S2 DTR/DSR signaling
AT&V	View Configuration
AT&W	Write Configuration (Stores active configuration in memory)
AT&Sxxx	Read S_Register xxx
AT&Sxxx=yyy	Set S_Register

Table 2.2: AT Commands

2.2.3. Parameters: S_Register

The S_register are the operating parameters or characteristics of the module. Table 2.3 at page 26 shows the different S_Registers and the last part of MHX920 command configu-

[8] In order to send more than one command line, wait for a response before entering the AT prefix at the start of the next command line.

ration; default setting s are emphasised with bold characters.

2.2.4. Configure modules

The MHX 920 default configuration is not exactly the configuration desired in this project. The default configuration and the useful one do not differ in main parameters except from Network type.

Configuring modules is required to set up the Point to Point Network. One module has to be declared as master and the other one as slave. This Network is configured by following next steps[9]:

1. Move to command mode[10]
2. Key “AT&F6” for Master module in PtP Network Key “AT&f7” for Slave module in PtP Network
3. Type “AT&/W”. Save current configuration
4. Type “ATA” or “ATO”

2.3. Servo 8 Torque Board

The “Servo 8 Torque Board” is a RS232 hobby servo controller. This servo controller is used in this case because it allows to control servo motors via serial port. The servo torque board is able to control up to 8 servos at the same time. Economy, purchase facility and RS232 connectivity have been the main factors to select this board as the best option.

The torque board measures torque by analysing the servo. When a command is sent to a servo to move to a certain position, the servo takes the difference between its current position and the commanded one. This difference is called the error. In attempting to drive the error to zero, the servo will apply power to the motor to minimise the error.

2.3.1. Commands

Each command is a sequence of bytes sent to the servo. The length of the sequence is four or five bytes, depending on the command. See Table 2.4 for details.

List of commands:

- **m** $\rightarrow$ Sends multiple position commands

[9] Each instruction must be followed by “<ENTER>”
[10] See instructions at section 2.2.1.2..

- **a** → Turns on servo and set position
- **a** 0 → Turns off servo
- **c** → Sets current position as home
- **h** → Goes to home position
- **w** → Sets width resolution
- **g** → Gets position
- **t** → Gets torque
- **s** → Sets module address
- **b** → Sets baud rate

One must be very careful when sending information to the Servo controller board, because it works with the Decimal Numeric code, not the ASCII one. To understand its operation mode, the difference is explained below.

Computers work by using the binary numeral system, or base-2 number system[11]. Instead of binary, there are also other numerical systems; e.g: the decimal[12] numerical system.

However, the traditional american and european keyboards use another type of code, which is called the ASCII[13] code.

This board configures the servo position from 0 to 255, with the decimal numeric system. **Nevertheless, take into account that this position is configured at decimal numeric system, not at ASCII code.** For example if number “8” is keyed, the real decimal number sent is 56. And character “{” corresponds to 123.

See Table C.1 for checking the relation between decimal numbers and ASCII code. Table 2.5 shows an example in ascii and decimal codes.

[11] *Binary Numerical code*: Is a numeral system that represents numeric values using two symbols, usually 0 and 1. More specifically, the usual base-2 system is a positional notation with a radix of 2. Owing to its straightforward implementation in electronic circuitry, the binary system is used internally by modern computers.

[12] *Decimal Numeric code*: Base ten or denary numerical system that has ten as its base. It is the most widely used numeral system, perhaps because humans have four fingers and a thumb on each hand, giving a total of ten digits over both hands.

[13] *ASCII*: ASCII American Standard Code for Information Interchange. It is a character encoding based on the English alphabet. ASCII codes represent text in computers, communications equipment, and other devices that work with text. Most modern character encoding, which support many more characters, have a historical basis in ASCII.

S_Register	Characteristic	Options
0	Auto Answer	
		0 Power up in command mode
		1 **Power up in data mode**
2	Escape Code	**“+”**
101	Operating mode	
		0 Master
		1 Repeater
		2 Slave
102	Serial Baud Rate	From 0 to 14
		0 230400
		7 **7: 9600**
		14 300
103	Wireless Link	
		1 115k
		2 **172k**
		2 230k
104	Network ID	From 0 to 4,000,000,000,000
105	Unit Address	From 2 to 65534
107	Encryption Phrase	**“Default”**
108	Output Power Level	20∼30dBm
110	Data Format	**8N1** (See sections 1.6.2.2.,1.6.2.3. and 1.6.2.4.)
113	Packet Retransmission	From 0 to 65535 **5**
118	Roaming	**Roaming enable**
119	Quick enter command	
		0 **Enable**
		1 Disable
123	RSSI Reading	
133	Network type	
		0 PtM
		1 PtP
		2 P2P (Peer-to-Peer)
140	Destination Address	From 1 to 65535
141	Repeater Existence	**0 No repeater**
142	Serial Channel Mode	
		0 **RS232**
		1 Half Duplex RS485
		2 Full Duplex RS485
143	Sleep mode	From 0 to 4. **0 No sleep**
144	Sleep Duration	From 0 to 65535 s
145	Awake Timeout	From 0 to 65535 s
150	Quick Sync Mode	**0 Normal Sync**
151	Quick Sync Timeout	From 100 to 6553. **200**
217	Protocol type	**0 Transparent**
237	Sniff Duration	From 1 to 255. **10**

Table 2.3: S_Registers

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Header	**Module Address**	**Servo Address**	**Command**	**Data (optional)**
“>”	“1 - 8”	“1 - 8”	“Command”	“optional”

Table 2.4: Command format

Byte #	1	2	3	4	5
ASCII	“>”	“1”	“2”	“a”	“–”
Decimal	62	49	50	97	255

Table 2.5: Command format example

CHAPTER 3. LIBRARIES

The project has developed eight different libraries which define the basics to establish wireless data transmission, NMEA reading, the servomotor control, image loading and plotting:

- Serial Port
- MHX
- AT
- S_Register
- NMEA
- Image
- Datum
- Servo

3.1. Serial Port

3.1.1. Objective

The MHX 920 modules are used to allow wireless data transfer. These wireless modules, as it has been said, use a RS232 interface, so it is necessary to establish serial communication with the module. This is why the serial port class has been added to the program. Using this class the program can send/receive data to/from the serial port.

3.1.2. Libraries

The serial port configuration implies configuring the computer internals. Accessing the serial port needs extra libraries that have been included. These libraries are:

- `#include <sys/types.h>`
- `#include <sys/stat.h>`
- `#include <sys/ioctl.h>`
- `#include <fcntl.h>`
- `#include <unistd.h>`

- `#include <termios.h>`
- `#include <errno.h>`
- `#include <sys/types.h>`

Other libraries which are more common and not specific to serial port are:

- `#include <iostream>`
- `#include <fstream>`
- `#include <sstream>`
- `#include <string>`
- `#include <cstring>`
- `#include <stdio.h>`
- `#include <stdlib.h>`

3.1.3. Class

The serial port class includes the next functions:

- `Serial();` Constructor that set up the initial values
- `~Serial();` Deconstructor that destroys serial class
- `int initPort();` Initialises the Serial Port. Its tasks are:
  - Try to open the port with r/w access.
  - Configure port settings.
  - Return -1 if init has failed and any error has occurred
- `string getData();` Reads the buffer by reading until n characters are read. Returns a string with the characters read.
- `int sendData(string data);` Writes to the buffer and sends a complete string. Receives the string that has to be sent. Returns the number of bytes sent.
- `char getChar();` Reads a byte from the buffer. Returns the char read. It is configured in ASCII code.
- `int sendChar(char data);` Writes to the buffer and sends a byte. Receives the char to be sent. Returns the number of bytes sent, which is one.
- `void flushPort();` Clears all data that is currently pending in either the transmit or receive buffers.

- `string get_line();` Given that port reading is faster than the speed NMEA sentences are sent, it is necessary to read the buffer byte by byte. This means that the program reads by using the getChar() function. However, it is not confortable to receive the NMEA sentence byte per byte and this function reads bytes until the NMEA sentence is complete. Returns a string with all the bytes read.
- `void closePort();` The serial port communication is closed and the port freed.

3.2. MHX

3.2.1. Objective

The MHX920, which is the wireless module used to transmit data, needs to be configured to be able to work.

In addition, the MHX has two different modes of operation and it is necessary to work in both modes and move from one to another. The MHX establishes the possibility to determine the MHX920 mode of operation and to communicate with the Serial Port.

3.2.2. Libraries

Libraries included in MHX are:

- `#include <cstring>`
- `#include <sstream>`
- `#include <string>`
- `#include <iostream>`
- `#include <stdio.h>`
- `#include ''AT.h''` This class, which is explained after the MHX one at section 3.3., allows the user to configure the MHX920 when it is in command mode.
- `#include ''serial.h''` As the program uses the serial port, and the communication will be done by the MHX920, the serial library has been included in this section. It is also possible to use the serial library separated from MHX.

3.2.3. Class

To control the MHX920 and their modes of operation, the MHX class needs to include the objects:

- `Serial port;`
- `AT com;`

The functions related to MHX class are:

- `MHX();` Constructor that sets up the initial values. The MHX has its own initial configuration. This configuration is the same as the one used in our case - for example baud rate 9600 - so there are no initial values declared. This constructor has been created if some day it is necessary to determine new ones.
- `~MHX();` Desconstructor that destroys MHX class.
- `int init_mhx();` Initialises the MHX values and the serial port object. Returns 0 when it works.
- `void mode_command();` Initialises or changes to mode command. Defines the MHX920 as master or slave and stores the configuration established.
- `void mode_data();` Initialises or changes to data mode.
- `void mode_change();` Changes from one mode to the other. Not used as it is not useful for our application.
- `int close_mhs();` Closes the MHX920 and memory is freed.
- `int char_length(char* data);` Counts the number of bytes included into a pointer char. The original serial port worked with pointer char and it was modified to string. The function has been left if necessary for future applications that need to return to the original.
- `string ch_to_s(char *data);` Receives a pointer char. Returns the same information into a string.
- `char* s_to_ch(string data);` Receives a string. Returns the same information into a pointer char.

3.3. AT

3.3.1. Objective

As it has been said before, the MHX920 needs to be configured to be able to work. To define and modify the different MHX920 properties, class AT has been created. These variables affect the module operation in data mode and the transition between data and command modes.

3.3.2. Libraries

The libraries included in AT class are:

- `#include <iostream>`
- `#include <stdio.h>`
- `#include ''S_Register.h''`. S_Register is a class created to establish the last configuration values. See section 3.4.

3.3.3. Class

The AT class does not have functions defined to work and modify data received. It has some variables which work as the different MHX920 commands.

- `char* F;` → Load factory default configuration
- `char* D;` → Data Terminal Ready (DTR)
- `char* K;` → Handshaking
- `char* S;` → Data Set Ready (DSR)
- `char* Dxx;` → Dial
- `char* I;` → Identification
- `char* V;` → View configuration
- `char* W;` → Write configuration to memory
- `char* Sxx;` → Read S Register value
- `char* Sxy;` → Set S Register value

It includes one constructor that sets up the initial values and the set and get functions as the variables are defined as private.

3.4. S_Register

3.4.1. Objective

Once the command mode is working and the different AT values defined, it is possible to work with the S_Register values. The S_Registers described in this section affect the operating characteristics of the module:

3.4.2. Libraries

S_Register does not need more than the `#include <stdio.h>` library. This library is needed to allow input and output operations.

3.4.3. Class

The S_Register class, as AT class does not have functions defined to work and modify data received. It has some variables to work with as MHX 920 properties.

- `char* S0` → Auto Answer
- `char* S2` → Escape code
- `char* S101` → Operating mode
- `char* S102` → Serial Baud Rate
- `char* S103` → Wireless Link Rate
- `char* S104` → Network ID
- `char* S105` → Unit address
- `char* S107` → Encryption Phrase
- `char* S108` → Output Power Level
- `char* S110` → Data Format
- `char* S113` → Packet Retransmission
- `char* S118` → Roaming
- `char* S119` → Quick Enter to Command
- `char* S123` → RSSI Reading (dBM)
- `char* S133` → Network Type
- `char* S140` → Destination Address
- `char* S141` → Repeater Existence
- `char* S142` → Serial Channel Mode
- `char* S143` → Sleep Mode
- `char* S144` → Sleep Duration
- `char* S145` → Awake Timeout
- `char* S150` → Quick Sync Mode

- `char* S151` → Quick Sync Timeout
- `char* S217` → Protocol type
- `char* S237` → Sniff Duration

Like the AT class, the S_Register includes one constructor that sets up the initial values and the set and get functions as the variables are defined as private.

3.5. NMEA

3.5.1. Objective

The National Marine Electronics Association has created an interface that allows communication between electronic equipments. This type of communication between equipments is the one used in this case to communicate a GPS receiver with a computer.

The information is sent by the GPS and received by the computer as complete NMEA sentences. The NMEA class is the one that recognises the sentences sent and classifies the information and checks the “checksum value”.

3.5.2. Libraries

The libraries used in the NMEA class are:

- `#include <iostream>`
- `#include <fstream>`
- `#include <sstream>`
- `#include <string>`
- `#include <cstring>`
- `#include <vector>`

3.5.3. Class NMEA

The NMEA includes inheritance. There is a basic NMEA class and the other different NMEA classes are created by inheritance from NMEA.

The NMEA class includes the values and functions which are common to all NMEA sentences. It has only three elements which are:

- `string GP` → The NMEA sentence
- `string name` → The NMEA name that allows us to know the type of information received
- `string sum` → The checksum value

The “public” functions are:

- `NMEA()`; Constructor that sets up the sentence as “NULL”.
- `virtual ~NMEA() ;` Virtual deconstructor.
- `virtual string send_nmea() = 0;` This is a pure virtual method. This means that any object created from NMEA **must** implement a “string send_nmea()” function. Returns a NMEA sentence.
- `virtual void receive_nmea(string GPa);` Virtual function that receives a GPa string which is a complete NMEA sentence.
- `virtual void modify(string op, string date) = 0;` Pure virtual method that modifies one value from any subclass created from NMEA.
- `string get_name();` Returns the NMEA sentence name.
- `void space();` Substitutes spaces for commas.

The “protected” functions are:

- `int checksum();` Checks a NMEA sentence to know if the checksum corresponds to the information sent.
- `string calculate_checksum(string GPa);` Receives a NMEA sentence without checksum and calculates it in order to send the NMEA sentence complete, with its corresponding checksum.

3.5.4. Derived classes

Defined the NMEA common methods in point3.5.3., the program uses inheritance to develop the NMEA derived classes. As no all NMEA sentences are sent by GPS receiver, there is a list of classes implemented in this project. List of NMEA sentences defined:

- GGA → Fix information
- GSA → Overall Satellite data
- GSV → Detailed Satellite data

- WPL → Waypoint Location information
- AAM → Waypoint Arrival Alarm
- APB → Auto Pilot B sentence
- RMC → Recommended minimum data for gps
- GLL → Lat/Lon data
- VTG → Vector track and Speed over the ground
- ZDA → Date and Time

We will show as example the GGA NMEA sentence. The other sentences are equal to GGA but modifying their field values.

NMEA_GGA:

- `NMEA_GGA();` Constructor which sets up the name as "GGA".
- `void set();` Reads a GGA sentence and organises the values in their respective strings.
- `string send_nmea();` From the created NMEA_GGA object, returns a new string by adding the values in their order and the checksum.
- `void modify(string op, string date);` Created to modify any of the sentence values once it is received, except the name and checksum.

  In this project this method is not used because the GPS receiver only transmits NMEA sentences, but it will become useful for new applications with GPS receivers able to receive information sent as NMEA sentences.

The list of NMEA_GGA values declared all of them as `string` is:

- `UTC` → Fix taken at UTC (hhmmss)
- `lat_val` → Latitude
- `lat_dir` → Latitude direction
- `lon_val` → Longitude
- `lon_dir` → Longitude direction
- `fix` → Fix quality
- `sat_num` → Number of satellites being tracked
- `dilution` → Horizontal dilution of precision
- `alt` → Altitude

- `alt_m` → Altitude unit (m)
- `wgs_height` → Height of geoid above WGS84 ellipsoid
- `wgs_height_m` → Height unit (m)
- `time` → Time in second since last DGPS update
- `dgps_id` → DGPS station ID number

Finally, to determine the last important particularities of this class, the enumerations TypeNMEA and enum_nmea have to be described with their respective functions::

- `TypeNMEA sentence_nmea(string GP);` Receives the NMEA sentence and selects which type of sentence it is.
- `enum_nmea nmea_val(string name);` While receiving the variable name, it selects this variable and allows the user to modify its value.

3.6. Image

3.6.1. Objective

The libraries created are a base to work with, amd it can be used to develop different programs to work with GPS. As a GPS receiver gives position as latitude and longitude, it is possible to show the GPS position in a map.

Knowing latitude and longitude will be enough to represent the position in a map. This library loads an image, in this case a map, and at the same time plots the position in the map, refreshing the window.

3.6.2. Libraries

The libraries included in the Image class are:

- `#include <stdio.h>`
- `#include <stdlib.h>`
- `#include <string.h>`
- `#include <iostream>`
- `#include <SDL.h>`
- `#include <SDL_ttf.h>`

- `#include <SDL_mixer.h>`
- `#include <SDL_image.h>`
- `#include ''datum.h''` The datum.h library is defined at section 3.7.. This library develops the datum change from WGS84 to ED50.

The SDL libraries are the ones that load and show images in C++. These libraries need some changes at the compilation options. When working with the "Anjuta" IDE[1], you have to:

1. Go to "Opciones del compilador y enlazador"
2. Go to "Biblioteca"
3. Insert:
   - SDL_image
   - SDL_ttf
   - SDL_mixer
4. Go to "Opciones"
5. Write in "CFLAGS": sdl-config –cflags‘ ‘sdl-config –libs‘
6. Click "Close"

Other IDE's should have a similar configuration procedure.

3.6.3. Class

The Image class includes SDL objects and the Datum class. As the objective of Image is to show a map and to represent some pixels inside it, this library has very few methods:

- `int load_map();` Loads the map and returns -1 if something did not work.
- `int close_map();` Closes the map. The program or representation is finished and it closes the image.
- `void put_pixel(int x, int y, Uint32 pixel);` Puts a black pixel/cross in position (x,y).
- `void new_pixel(float lat, float lon);` Receives the latitude and longitude in WGS84 and changes to ED50. Calls "void put_pixel" and represents the new pixel.

[1] *Anjuta IDE*: is an Integrated Development Environment for C and C++ on GNU/Linux. It has been written for GTK+/GNOME and features a number of advanced programming facilities.

3.7. Datum

3.7.1. Objective

The GPS uses the WGS84 geodetic system, while european maps work with the ED50. In order to represent positions received by a GPS in a map, it is necessary to change latitude and longitude from WGS84 to ED50. Once the position is in ED50 system, the position has to be related with the pixels of the image.

Therefore, the datum library changes positions from WGS84 to ED50, and then it establishes the relation between ED50 and the pixel matrix. This way, after receiving a WGS84 position the library returns the respective pixel position.

3.7.2. Libraries

The libraries used in Datum are:

- `#include <iostream>`
- `#include <stdio.h>`
- `#include <cstring>`
- `#include <sstream>`
- `#include <stdio.h>`
- `#include <string>`
- `#include <math.h>`: This library is used to reproduce more specific math operations.

3.7.3. Class

The Datum class is peculiar because it has been created to save all the different values used to represent position. It has as variables:

- `float wgs_lat / wgs_lon / wgs_alt`: Position in datum WGS84
- `float ed_lat / ed_lon / ed_alt`: Position in datum ED50
- `int ph / pv`: Position in horizontal and vertical pixels.

The process to find the ED50 Geodesic coordinates is always the same, but it has been separated in different methods to relate one method to one process pass:

- `void radian()` Changes latitude and longitude from degrees to radians.
- `void wgs_ed();` Calculates latitude and longitude in ED50
- `void map();` Calculates the pixel position
- `pixel(float lat, float lon);` Does all the processing by calling the other methods.
- `int p_ph();` Returns horizontal pixel position.
- `int p_pv();` Returns vertical pixel position.

3.8. Servo

3.8.1. Objective

In order to be able to control servo motors using a computer, the project uses the “Serial Servo Controller” board.

This board works with really simple commands, but it is necessary to write them in binary configuration. The servo library has been created to send this commands using the keyboard.

3.8.2. Libraries

The only library necessary to add to Servo class is `#include<stdlib.h>`.

3.8.3. Class

The Servo class, which is created to control servos, is defined by these methods:

- `void forward();` Sends the “forward” order.
- `void reverse();` Sends the “reverse” order.
- `void left();` Sends the “left” order.
- `void right();` Sends the “right” order.
- `void stop();` Stops any movement.
- `void stop_fr();` Stops from moving f-r[2].

[2] Forward-reverse.

- `void stop_lr();` Stops from turning l-r[3].
- `void instruction_fr();` Sends the servo controller f-r port position[4].
- `void instruction_lr();` Sends the servo controller l-r port position[5].

[3] Left-right.
[4] The servo controller controls 8 different ports. The f-r port is the 6th one.
[5] The l-r port is the 3rd one.

CHAPTER 4. EXPERIMENTS

4.1. GPS position transmission

4.1.1. Introduction

This experiment is the result of joining all systems. The experiment #1 starts up the data transmission system using the GPS receiver, the MHX 920 modules and the laptop.

The GPS receiver transmits NMEA sentences. These sentences arrive to the MHX 920 slave module. The MHX 920 slave module then transmits this information to the master module at the base station. This data is transmitted to the laptop where it is classified by the program. Latitude and longitude values are used to print information upon a Castelldefels map.

It is very important to split the process in parts in order to be sure that each part is working properly and there are no mistakes. This is fundamental for this type of experiments where lots of different elements of hardware and software are related. To achieve the experiment success, it has been carried out by dividing the process in sub-parts. If there is any problem in transmitting, working this way allows the user to recognise easily where the problem is.

4.1.2. Test: Serial Port

First of all, serial communication between devices was necessary. This test was the first one, because lack of serial communication does not allow other tests. To test communication, a null modem cable has been used.

This null modem cable allows direct communication between computers. Two computers were connected by null modem cable and using `minicom`[1] program, transmission was achieved.

It was necessary to establish the same serial port configuration, which is showed at section 1.6.3. This test proved serial port communication between devices, and allowed other tests to be carried out.

4.1.3. Test: ANTARIS SBEKit

Secondly, it is necessary to know if the ANTARIS GPS receiver is sending any information and to verify that it is correct. The process has been done by using the U-Center ANTARIS Edition GPS Evaluation Software[2]. The experiment was done by connecting the ANTARIS

[1] *Minicom*: A friendly free software serial communication program. Similar to Hyperterminal in Windows.

[2] This sofware is included in the ANTARIS SBEKit. It does not work in LINUX so it has been proved in Windows XP.

SBEKit to a laptop by serial port[3].

ANTARIS SBEKit and the laptop were placed at the shopping cart used for safely transporting the equipment, and connected by serial port. As said before, GPS receiver was controlled by U-center program.

Moving the shopping cart, position received at computer was received once each 5 or 10 minutes. Something from ANTARIS SBEKit configuration did not allow continuous position transmission. As the ANTARIS SBEKit has been used in other TFCs, it was reset to default configuration and EKF[4] was disabled. After doing this, GPS receiver started to transmit its position each second.

4.1.4. Test: MHX 920 modules

The data transmission system must be developed between a fixed base station and a mobile one. This is the MHX 920 modules function. To achieve this purpose the modules needed to be tested before trying the complete experiment.

The MHX 920 modules were configured as Point-to-Point transmission by command mode. One module was also configured as master and the other as slave[5]. The master module would remain at base station and the slave would be the mobile one.

This test was executed by connecting a PC and a laptop to each of the MHX 920 modules. The master module was connected to a PC at the base station situated at the EPSC Aeronautical Department. The mobile one was placed at the shopping cart and moved all around the university.

To prove the wireless data transmission system, it was necessary to sent information dynamically. Tests were done connecting and sending dynamic information to a laptop. By using minicom, data transmission by serial port was assured. The PC was sending UTC time continuously each second. At the laptop, the UTC time information was received.

For a fifteen minutes-long test, no more than 3 seconds were lost; the exception is when the laptop was inside the lift, where communication was interrupted.

4.1.5. Test: Software

In addition, the new software must be tested in order to know if libraries are working correctly, both alone and together. A basic main program has been developed to use the libraries created.

This program receives information, classifies it, shows latitude and longitude values and plots on a Castelldefels map the GPS receiver position. It has been proved in two ways:

[3] Actually, given that the laptop did not have a serial port, an USB-Serial Port adapter was used.

[4] *EKF*: Enhanced Kalman Filter algorithm. It provides precise navigation in locations with no or impaired GPS reception.

[5] See section 2.2.4.

- Connecting laptop and ANTARIS directly by serial port
- Connecting laptop and ANTARIS using the MHX 920 modules.

The first test was done after ANTARIS SBEKit test[6] was successfully achieved. Laptop's operating system was changed from Windows XP to Ubuntu Linux. At Ubuntu, the new sofware was tried and the result is shown in Figures 4.1 and 4.2.

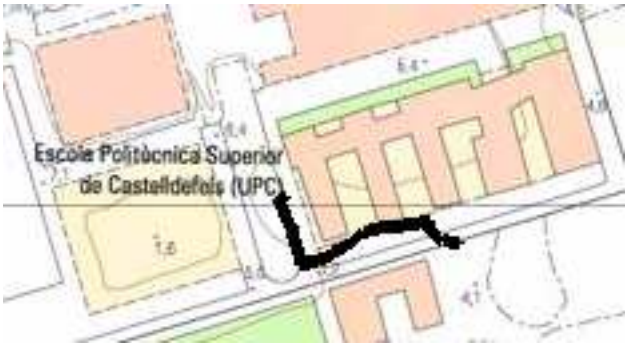

Figure 4.1: First software trial. Halfway the experiment

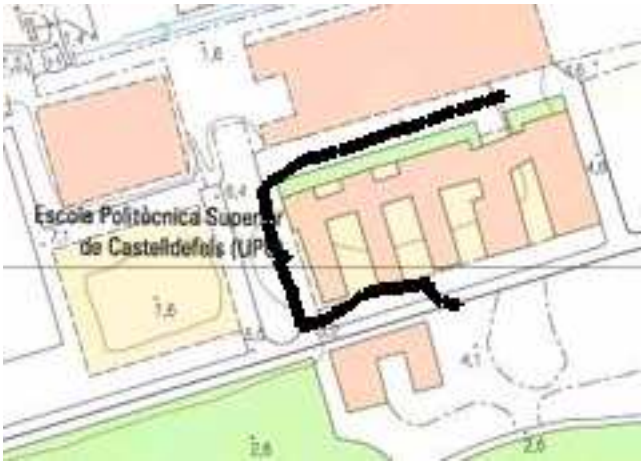

Figure 4.2: First software trial.

The second trial was done by connecting everything. As it uses and connects all the elements, it has been specified at section 4.1.6..

4.1.6. GPS position transmission

Finally, this trail assembles all the different hardware and software elements. It has been carried out after all the other tests have been accomplished and the elements worked properly.

The ANTARIS SBEKit and the MHX 920 slave module were connected by a null modem cable and placed in the shopping cart. They formed the mobile station. The master module, connected to the laptop, formed the fixed base station. The fixed station was installed

[6] See 4.1.3. at page 43

at EPSC Aeronautical Department. With these two stations, the experiment was ready to be carried out.

As soon as program was executed, the transmission of information was correct, mobile station latitude and longitude figures were shown in the terminal, and position was plot in the map. The last latitude and longitude values were (See Figure 4.3):

- Latitude: `4116.504700`
- Longitude: `159.210960`
- Pixel: `(642,219)`

```
Lat: 4116.504700
Lon: 159.210960
```

Figure 4.3: Last latitude and longitude

Results are shown in Figure 4.4.

Figure 4.4: Mobile station position

4.2. Servo control transmission

4.2.1. Introduction

This experiment tries to develop a system which allows the automatic movement of mobile station by remote control. The experiment #2 creates a remote controlled mobile station using the MHX 920 modules. It uses the MHX 920 modules, the Servo controller board and the laptop.

The laptop sends movement information to the servo controller board. Data is transmitted via the MHX 920 modules.

4.2.2. Test: Servo movement

First of all, to develop a radio control mobile station, a propulsion system was needed to push the station. In this case the station movement was achieved by using a radio-controlled car with servomotors. First test done was to ensure that servos moved correctly.

This test was carried out using the radio control car's radio transmitter.

4.2.3. Test: Servo controller board

Secondly, it is necessary to be able to send data information to servos to control them, which is the servo controller board function. This test proves the servo controller board.

The board was connected directly to the laptop, and the servos were connected to the board. By using minicom, the laptop was able to transmit data by serial port. Given that the serial port test was done at experiment #1, it was not necessary to realise this test again.

In order to establish the physical connections order, first it is necessary to establish the serial communication between laptop and board; then connect the servo wires to the servo board, and finally apply power[7] to the servo board.

Opening minicom, serial communication was established. Servo controller board default configuration was at baud rate 19200, so the default configuration was changed to baud rate 9600. This configuration allows the program to continue using the 9600 baud rate and do not modify the MHX 920 baud rate. The idea was to establish a standard configuration for all the elements.

Simple orders in ASCII code, like "$>$11a9", were issued and the servos moved to that position[8].

4.2.4. Test: Software

In addition, another main program was programmed to control servos. This software uses the same libraries as experiment #1 but does not work with any image or NMEA sentence. It needs the MHX 920, serial port and servo libraries. This basic main program includes the libraries created and allows user to communicate with servomotors via servo controller board.

The program has five different orders, which are:

[7] The servo controller board has been designed to share power between board and servos.

[8] Note that position sent was not 9, but 57. See Table C.1

- **w** → Forward movement
- **s** → Reverse movement
- **a** → Left movement
- **d** → Right movement
- **q** → Exit the program

User introduces any of this orders and program sends the data necessary to control servos.

4.2.5. Radio controlled mobile station

Finally, this trail assembles all the different hardware and software elements. It has been carried out after all the other tests have been accomplished and the elements worked properly. Some tests have not been done in experiment #2 because they were the same as others done before, such as serial communication.

The Servo controller board and the MHX 920 module were connected by a null modem cable and placed at the radio control car. They are the new mobile station controlled by radio control, but our own radio control. The master MHX 920 module has been connected to the laptop and forms the base station. With the two stations created, the experiment was able to be carried out.

WHen the program was run, the base station was controlling by wireless communication the servomotors position and the movement of the remote controlled car. Servos reacted to the different orders sent. See the mobile station at Figure 4.5.

Figure 4.5: Remote controlled car

CONCLUSIONS

The successfully fulfilment of the experiments realised, supposes the achievement of the primary objective initially raised. It was also possible to achieve the secondary objectives while developing the main one. The wireless data communication system formed by the two MHX 920 modules, in addition to the developed software tools, have given the opportunity to achieve all secondary objectives.

Establishing a sound theoretical basics has make possible to develop this project. The most complex a project is, the most important the theorical foundations are. This basis have facilitated the hardware configuration and use as well as software design and development.

Setting up the hardware elements used and the software tool have allow the experiments achievement. These experiments prove the utility of the software tool.

However, the main importance of this project is its usefulness as a software tool base. It can act as a base to develop and implement lots of new applications. For instance, the remote controlled mobile station may informs its own position and may also report climatological data. Nevertheless, this possibilities will only be able while having another MHX 920 module or a serial multiplexer adaptor that allows communication between more than two devices.

APPENDIX A. GPS ANNEXES

A.1. GPS Annexes

A.1.1. Calculating position

As said in 1.2.2., in order for a GPS receiver to determine its position, it has to receive signals from four different satellite to enable it to calculate signal transit time ($\Delta t_1 \ldots \Delta t_4$).

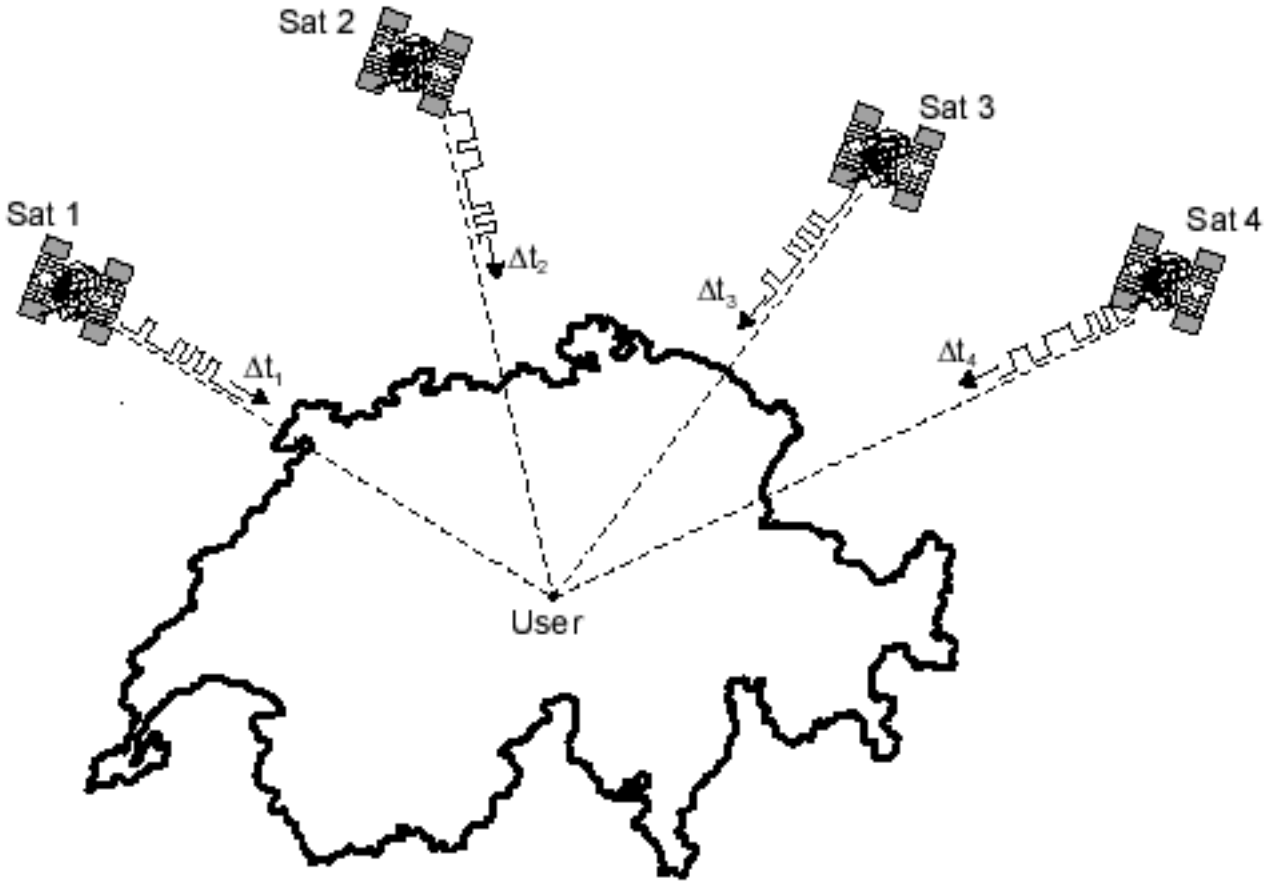

Figure A.1: Four Satellite signals must be received

Calculations are done in a Cartesian, three-dimensional coordinate system with a geo-centric origin. The range of the user from the four satellites R1, R2, R3 and R4 can be determined with the help of signal transit times $\Delta t_1, \Delta t_2, \Delta t_3, \Delta t_4$ between the four satellites and the user. As the locations X_{sat}, Y_{sat} and Z_{sat} of the four satellites are known, the user coordinates can be calculated.

Distances measured between each satellite and the receiver are not the correct distances since there are inaccuracies in time measurement, among others. That is the reason

because they are called Pseudo-distances or Pseudo-ranges[1]. The computation has to be done in Cartesian format, in a three-dimensional coordinate system with geocentric origin.

$$\Delta t_{measured} = \Delta t + \Delta t_0 \tag{A.1}$$

$$PSR = \Delta t_{measured} * c = (\Delta t + \Delta t_0) * c \tag{A.2}$$

$$\mathbf{PSR} = R + \Delta t_0 * c \tag{A.3}$$

Where:

- R: True range from the satellite to the user.
- c: Speed of light.
- Δt: Signal transit time from the satellite to the user.
- Δt_0: Difference between the satellite clock time and the user clock time.
- PSR: Pseudo-range.

The distance R from the satellite to the user can be calculated in a Cartesian system as follows:

$$R = \sqrt{(X_{sat} - X_{user})^2 + (Y_{sat} - Y_{user})^2 + (Z_{sat} - Z_{user})^2} \tag{A.4}$$

And therefore:

$$PSR = \sqrt{(X_{sat} - X_{user})^2 + (Y_{sat} - Y_{user})^2 + (Z_{sat} - Z_{user})^2} + c\Delta t_0 \tag{A.5}$$

In order to determine the four unknown variables ($\Delta t_0, X_{user}, Y_{user}$ and Z_{user}), four independent equations are necessary. The following formula is valid for the four satellites (i = 1 . . . 4):

$$PSRi = \sqrt{(X_i - X_{user})^2 + (Y_i - Y_{user})^2 + (Z_i - Z_{user})^2} + \Delta t_0 * c \tag{A.6}$$

A.1.2. Linearisation of the equation

The four equations under A.6 produce a non-linear set of equations. In order to solve the set, the root function is first linearised according to the Taylor theorem, and only the linear terms will be used.

[1] *PSR*: Pseudo-Range.

Taylor theorem:

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!} * \Delta x + \underbrace{\frac{f''(x_0)}{2!} * \Delta x^2 + \ldots}_{\text{Small values are rejected}} \tag{A.7}$$

Taylor theorem in a three-dimensional system:

$$f(x,y,z) = f(x_0,y_0,z_0) + (\frac{\partial f}{\partial x})\Big|_0 * \Delta x + (\frac{\partial f}{\partial y})\Big|_0 * \Delta y + (\frac{\partial f}{\partial z})\Big|_0 * \Delta z \tag{A.8}$$

In order to linearise the equations system, an arbitrarily estimated value ($x_{est}, y_{est}, z_{est}$) must be known.

Instead of calculating the true position (x,y,z) from the estimated position, it is necessary to calculate the variables $\Delta x, \Delta y$ and Δz:

$$\begin{aligned} X_{true} &= X_{est} + \Delta x \\ Y_{true} &= Y_{est} + \Delta y \\ Z_{true} &= Z_{est} + \Delta z \end{aligned} \tag{A.9}$$

By the way, distance (R_{est}) between receiver estimated position and satellites is:

$$R_{est_i} = \sqrt{(X_i - X_{est})^2 + (Y_i - Y_{est})^2 + (Z_i - Z_{est})^2} \tag{A.10}$$

Applying Taylor theorem to formula A.6:

$$PSR_i = R_{est_i} + (\frac{\partial R_{est_i}}{\partial x}) * \Delta x + (\frac{\partial R_{est_i}}{\partial y}) * \Delta y + (\frac{\partial R_{est_i}}{\partial z}) * \Delta z + \Delta t_0 * c$$

Solving the partial derivatives:

$$\frac{\partial R_{est_i}}{\partial x} = [\frac{1}{2 * \sqrt{(X_i - X_{est})^2 + (Y_i - Y_{est})^2 + (Z_i - Z_{est})^2}}] * 2 * (X_i - X_{est}) * (-1) = \frac{X_{est} - X_i}{R_{est_i}}$$

$$\frac{\partial R_{est_i}}{\partial y} = \ldots = \frac{Y_{est} - Y_i}{R_{est_i}}$$

$$\frac{\partial R_{est_i}}{\partial z} = \ldots = \frac{Z_{est} - Z_i}{R_{est_i}}$$

Once solved all the partial derivatives the following formula is obtained:

$$PSR_i = R_{est_i} + (\frac{X_{est} - X_i}{R_{est_i}}) * \Delta x + (\frac{Y_{est} - Y_i}{R_{est_i}}) * \Delta y + (\frac{Z_{est} - Z_i}{R_{est_i}}) * \Delta z + \Delta t_0 * c \qquad \text{(A.11)}$$

A.1.3. Solving the equations

After transposing the four satellites' equations (for i = 1 . . . 4), the four variables ($\Delta x, \Delta y, \Delta z$ and Δt_0) can now be solved according to the rules of linear algebra:

$$\begin{vmatrix} PSR_1 - R_{est_1} \\ PSR_2 - R_{est_2} \\ PSR_3 - R_{est_3} \\ PSR_4 - R_{est_4} \end{vmatrix} = \begin{pmatrix} \frac{X_{est}-X_1}{R_{est_1}} & \frac{Y_{est}-Y_1}{R_{est_1}} & \frac{Z_{est}-Z_1}{R_{est_1}} & c \\ \frac{X_{est}-X_2}{R_{est_2}} & \frac{Y_{est}-Y_2}{R_{est_2}} & \frac{Z_{est}-Z_2}{R_{est_2}} & c \\ \frac{X_{est}-X_3}{R_{est_3}} & \frac{Y_{est}-Y_3}{R_{est_3}} & \frac{Z_{est}-Z_3}{R_{est_3}} & c \\ \frac{X_{est}-X_4}{R_{est_4}} & \frac{Y_{est}-Y_4}{R_{est_4}} & \frac{Z_{est}-Z_4}{R_{est_4}} & c \end{pmatrix} * \begin{vmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta t_0 \end{vmatrix}$$

Isolating $\Delta x, \Delta y, \Delta z$ and Δt_0:

$$\begin{vmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta t_0 \end{vmatrix} = \begin{pmatrix} \frac{X_{est}-X_1}{R_{est_1}} & \frac{Y_{est}-Y_1}{R_{est_1}} & \frac{Z_{est}-Z_1}{R_{est_1}} & c \\ \frac{X_{est}-X_2}{R_{est_2}} & \frac{Y_{est}-Y_2}{R_{est_2}} & \frac{Z_{est}-Z_2}{R_{est_2}} & c \\ \frac{X_{est}-X_3}{R_{est_3}} & \frac{Y_{est}-Y_3}{R_{est_3}} & \frac{Z_{est}-Z_3}{R_{est_3}} & c \\ \frac{X_{est}-X_4}{R_{est_4}} & \frac{Y_{est}-Y_4}{R_{est_4}} & \frac{Z_{est}-Z_4}{R_{est_4}} & c \end{pmatrix}^{-1} * \begin{vmatrix} PSR_1 - R_{est_1} \\ PSR_2 - R_{est_2} \\ PSR_3 - R_{est_3} \\ PSR_4 - R_{est_4} \end{vmatrix}$$

The solution of $\Delta x, \Delta y, \Delta z$ and Δt_0 is used to recalculate the estimated position ($X_{est-new}, Y_{est-new}, Z_{est-new}$) in accordance with equation A.12.

$$\begin{aligned} X_{est-new} &= X_{est} + \Delta x \\ Y_{est-new} &= Y_{est} + \Delta y \\ Z_{est-new} &= Z_{est} + \Delta z \end{aligned} \qquad \text{(A.12)}$$

The new estimated values can now be taken as the true position if estimated initial position was not very far away[2] from the true position.

[2] Hundreds of kilometres or more.

A.2. NMEA sentences

In this section we will show a set of tables containing the characteristics of the most common NMEA sentences.

Field	Description
$	Start of the data set
GP	Information originating from a GPS appliance
AAM	Data set identifier
A	Arrival Circle entered
A	Perpendicular passed
0.10	Circle radius
N	Circle radius units (Nautical miles)
WPTNME	Waypoint name
*	Separator for the checksum
04	Checksum for verifying the entire data set
<CR> <LF>	End of the data set

Table A.1: Description of the individual AAM Data Set Blocks

Field	Description
$	Start of the data set
GP	Information originating from a GPS appliance
APB	Data set identifier
A	Loran-C blink/SNR warning, general warning
A	Loran-C cycle warning
0.10	Cross track error distance
R	Steer Right to correct (L for left)
N	Cross-track error units Nautical miles (K for kilometers)
V	Arrival alarm circle
V	Arrival alarm perpendicular
011	Bearing, origin to destination
M	Magnetic bearing
DEST	Destination waypoint ID
011	Bearing, present to destination
M	Magnetic bearing
011	Heading to steer
M	Magnetic heading
*	Separator for the checksum
04	Checksum for verifying the entire data set
<CR> <LF>	End of the data set

Table A.2: Description of the individual APB Data Set Blocks

Field	Description
$	Start of the data set
GP	Information originating from a GPS appliance
GGA	Data set identifier
130305.0	UTC positional time: 13h 03min 05.0sec
4717.115	Latitude: 47° 17.115 min
N	Northerly latitude (N=north, S= south)
00833.912	Latitude: 8° 33.912min
E	Easterly longitude (E= east, W=west)
1	GPS quality details (0= no GPS, 1= GPS, 2=DGPS)
08	Number of satellites used in the calculation
0.94	Horizontal Dilution of Precision (HDOP)
00499	Antenna height data (geoid height)
M	Unit of height (Meter)
047	Height differential between an ellipsoid and geoid
M	Unit of differential height (Meter)
,,	Age of the DGPS data (in this case no DGPS is used)
0000	Identification of the DGPS reference station
*	Separator for the checksum
58	Checksum for verifying the entire data set
<CR> <LF>	End of the data set

Table A.3: Description of the individual GGA Data Set Blocks

Field	Description
$	Start of the data set
GP	Information originating from a GPS appliance
GSA	Data set identifier
A	Calculating mode
	A automatic selection between 2D/3D mode
	M manual selection between 2D/3D mode
3	Calculating mode (1= none, 2=2D, 3=3D)
13	ID number of the satellites used to calculate position
20	ID number of the satellites used to calculate position
11	ID number of the satellites used to calculate position
29	ID number of the satellites used to calculate position
01	ID number of the satellites used to calculate position
25	ID number of the satellites used to calculate position
07	ID number of the satellites used to calculate position
04	ID number of the satellites used to calculate position
,,,,,	Dummy for additional ID numbers (currently not used)
1.63	PDOP (Position Dilution of Precision)
0.94	HDOP (Horizontal Dilution of Precision)
1.33	VDOP (Vertical Dilution of Precision)
*	Separator for the checksum
04	Checksum for verifying the entire data set
<CR> <LF>	End of the data set

Table A.4: Description of the individual GSA Data Set Blocks

labeltab:gps-nmea-gsa

Field	Description
$	Start of the data set
GP	Information originating from a GPS appliance
GSV	Data set identifier
2	Total number of GVS data sets transmitted (1-9)
2	Current number of this GVS data set (1-9)
09	Total number of satellites in view
01	Identification number of the first satellite
52	Elevation (0° .... 90°)
187	Azimuth (0° ... 360°)
43	Signal-to-noise ratio in db-Hz (1-99)[3]
25	Identification number of the second satellite
25	Elevation (0° .... 90°)
074	Azimuth (0° ... 360°)
39	Signal-to-noise ratio in dB-Hz (1-99) null when not tracking)
07	Identification number of the third satellite
37	Elevation (0° .... 90°)
286	Azimuth (0° ... 360°)
40	Signal-to-noise ratio in db-Hz (1 ... 99, null when not tracking)
04	Identification number of the fourth satellite
09	Elevation (0° .... 90°)
306	Azimuth (0° ... 360°)
33	Signal-to-noise ratio in db-Hz (1 ... 99, null when not tracking)
*	Separator for the checksum
44	Checksum for verifying the entire data set
<CR> <LF>	End of the data set

Table A.5: Description of the individual GSV Data Set Blocks

Field	Description
$	Start of the data set
GP	Information originating from a GPS appliance
GLL	Data set identifier
4717.115	Latitude: 47° 17.115 min
N	Northerly latitude (N=north, S= south)
00833.912	Longitude: 8° 33.912min
E	Easterly longitude (E=east, W=west)
130305.0	UTC positional time: 13h 03min 05.0sec
A	Data set quality: A means valid (V= invalid)
*	Separator for the checksum
32	Checksum for verifying the entire data set
<CR> <LF>	End of the data set

Table A.6: Description of the individual GLL Data Set Blocks

Field	Description
$	Start of the data set
GP	Information originating from a GPS appliance
RMC	Data set identifier
130304.0	Time of reception (world time UTC): 13h 03 min 04.0 sec
A	Data set quality: A signifies valid (V= invalid)
4717.115	Latitude: 47° 17.115 min
N	Northerly latitude (N=north, S= south)
00833.912	Longitude: 8° 33.912 min
E	Easterly longitude (E=east, W=west)
000.04	Speed: 0.04 knots
205.5	Course: 205.5°
200601	Date: 20th June 2001
01.3	Adjusted declination: 1.3°
W	Westerly direction of declination (E = east)
*	Separator for the checksum
7C	Checksum for verifying the entire data set
<CR> <LF>	End of the data set

Table A.7: Description of the individual RMC Data Set Blocks

Field	Description
$	Start of the data set
GP	Information originating from a GPS appliance
VTG	Data set identifier
014.2	Course 14.2° (T) with regard to the horizontal plane
T	Angular course data relative to the map
015.4	Course 15.4° (M) with regard to the horizontal plane
M	Angular course data relative to magnetic north
000.03	Horizontal speed (N)
N	Speed in knots
000.05	Horizontal speed (Km/h)
K	Speed in km/h
*	Separator for the checksum
4F	Checksum for verifying the entire data set
<CR> <LF>	End of the data set

Table A.8: Description of the individual VTG Data Set Blocks

Field	Description
$	Start of the data set
GP	Information originating from a GPS appliance
WPL	Data set identifier
4717.115	Latitude: 47° 17.115 min
N	Northerly latitude (N=north, S= south)
00833.912	Longitude: 8° 33.912 min
E	Easterly longitude (E=east, W=west)
WPTNME	Waypoint name
*	Separator for the checksum
5D	Checksum for verifying the entire data set
<CR> <LF>	End of the data set

Table A.9: Description of the individual WPL Data Set Blocks

Field	Description
$	Start of the data set
GP	Information originating from a GPS appliance
ZDA	Data set identifier
130305.2	UTC time: 13h 03min 05.2sec
20	Day (00-31)
06	Month (1-12)
2001	Year
	Reserved for data on local time (h), not specified here
	Reserved for data on local time (min), not specified here
*	Separator for the checksum
57	Checksum for verifying the entire data set
<CR> <LF>	End of the data set

Table A.10: Description of the individual ZDA Data Set Blocks

APPENDIX B. GEODETIC AND CARTESIAN COORDINATES. DATUM

B.1. Coordinates: Geodesic and Cartesian

B.1.1. Geodetic to Cartesian coordinates

The Greek letter λ designates the longitude and the Greek letter ϕ designates the latitude. The letter h corresponds to the ellipsoidal height (not to be confused with altitude). It is defined in a geodesic reference system and can differ in altitude by several tens of metres.

$$\frac{w^2}{a^2} + \frac{z^2}{b^2} = 1 \tag{B.1}$$

where: $w^2 = x^2 * y^2$.

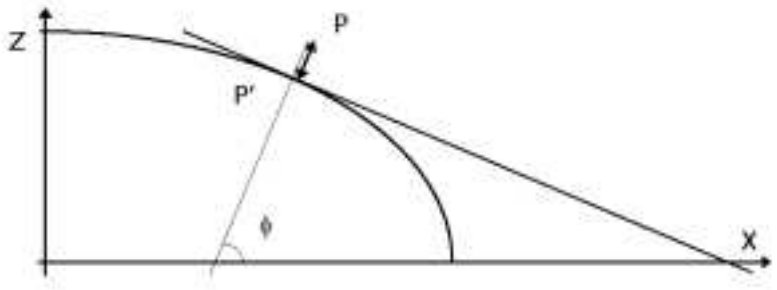

Figure B.1: Latitude ϕ and height h at an ellipsoidal surface

At Figure B.1, where X-axis represents w, equation B.2 is observed:

$$\tan\phi = -\frac{1}{\partial z/\partial w} = \frac{z}{w(1-e^2)} \tag{B.2}$$

Isolating z and w and expressing the ellipse depending on tan ϕ from B.2:

$$z = \frac{a(1-e^2)\sin\phi}{\sqrt{1-e^2\sin\phi}}$$
$$w = \frac{a}{\sqrt{1+(1-e^2\sin\phi)}} = \frac{a\cos\phi}{\sqrt{1-e^2\sin^2\phi}} \tag{B.3}$$

Since point P is determined by λ_1 and ϕ_1 (coordinates of its projection) and by h_1 (its height from the ellipsoid), the result is:

$$z_1 = z + h_1 \sin\phi_1 \tag{B.4}$$

$$w_1 = w + h_1 \cos\phi_1 \tag{B.5}$$

From Equation B.4, where z and w correspond to P' Cartesian coordinates, and knowing the eccentricity e_1 and the major semi-axis a_1, the transformation becomes:

$$\begin{aligned} z_1 &= \left(\frac{a_1(1-{e_1}^2)}{\sqrt{1-{e_1}^2 \sin^2\phi_1}} + h_1 \right) \sin\phi_1 \\ y_1 &= \left(\frac{a_1}{\sqrt{1-{e_1}^2 \sin^2\phi_1}} + h_1 \right) \cos\phi_1 \sin\lambda_1 \\ x_1 &= \left(\frac{a_1}{\sqrt{1-{e_1}^2 \sin^2\phi_1}} + h_1 \right) \cos\phi_1 \cos\lambda_1 \end{aligned} \tag{B.6}$$

B.1.2. Cartesian to Geodetic coordinates

Defined the Cartesian coordinate system, each point on the ellipsoidal surface answers to:

$$\frac{x^2}{a^2} + \frac{y^2}{a^2} + \frac{z^2}{b^2} = 1 \tag{B.7}$$

If x, y and z are known, it is possible to calculate λ, ϕ and h. As several values are unknown, the first approximation has been calculated by considering h as 1 m.

$$
\begin{aligned}
b &= \sqrt{(1-e^2)*a^2} \\
e' &= a\sqrt{e}/b \\
r &= \sqrt{x^2+y^2} \\
E^2 &= a^2-b^2 \\
F &= 54b^2z^2 \\
G &= r^2+(1-e^2)z^2-e^2E^2 \\
c &= \frac{e^4Fr^2}{G^3} \\
s &= \sqrt[3]{1+c+\sqrt{c^2+2c}} \\
P &= \frac{F}{3\Big((s+1)/(s-1)\Big)^2 G^2} \\
Q &= \sqrt{1+2e^4P} \\
r_0 &= -\frac{Pe^2r}{1+Q}+\sqrt{\frac{1}{2}a^2\Big(1+\frac{1}{Q}\Big)-\frac{P(1-e^2)z^2}{Q(1+Q)}-\frac{1}{2}Pr^2} \\
U &= \sqrt{z^2+(r+r_0e^2)^2} \\
V &= \sqrt{z^2(1-e^2)+(r-r_0e^2)^2} \\
z_0 &= \frac{b^2z}{aV}
\end{aligned}
\tag{B.8}
$$

From last values obtained at B.8, latitude, longitude and height (λ, ϕ and h) are:

$$
\begin{aligned}
h &= U\left(1-\frac{b^2}{aV}\right) \\
\phi &= \arctan\left(\frac{z+e'^2z_0}{r}\right) \\
\lambda &= \arctan\left(\frac{y}{x}\right)
\end{aligned}
\tag{B.9}
$$

B.2. Datum WGS84 to ED50

The measures realised by a GPS receiver cannot be plotted directly into a map, because current maps use another different ellipsoid. Position should be converted from WGS84 to ED50 LLH. GPS transmits LLH[1] coordinates and the transformation between WGS84 and ED50 must be done in Cartesian coordinates.

[1] *LLH*: Latitude, Longitude, Height.

The relation between WGS84 and ED50 is showed in section 1.4.4. at Table 1.3. In order to convert from Geodesic coordinates Datum WGS84 to Geodesic coordinates ED50, the process followed is:

Datum WGS84 LLH → Datum WGS84 XYZ → Datum ED50 XYZ → Datum ED50 LLH

Following the instructions to change coordinates systems and to pass from WGS84 to ED50, the complete process is:

Datum $\mathrm{WGS84}_1$ from Geodesic to Cartesian coordinates:

$$\begin{aligned} x_1 &= w_1 \cos\lambda_1 = \left(\frac{a_1}{\sqrt{1 - e_1^2 \sin^2 \phi_1}} + h_1 \right) \cos\phi_1 \cos\lambda_1 \\ y_1 &= w_1 \sin\lambda_1 = \left(\frac{a_1}{\sqrt{1 - e_1^2 \sin^2 \phi_1}} + h_1 \right) \cos\phi_1 \sin\lambda_1 \\ z_1 &= \left(\frac{a_1(1 - e_1^2)}{\sqrt{1 - e_1^2 \sin^2 \phi_1}} + h_1 \right) \sin\phi_1 \end{aligned} \tag{B.10}$$

Datum $\mathrm{WGS84}_1$ Cartesian coordinates to $\mathrm{ED50}_2$ Cartesian coordinates :

$$\begin{aligned} x_2 &= x_1 + \Delta_x \\ y_2 &= y_1 + \Delta_y \\ z_2 &= z_1 + \Delta_z \end{aligned} \tag{B.11}$$

Datum $\mathrm{ED50}_2$ Cartesian to Geodesic coordinates (See Equations: B.8):

$$\begin{aligned} \lambda_2 &= \arctan\left(\frac{y_2}{x_2}\right) \\ \phi_2 &= \arctan\left(\frac{z_2 + e'^2 z_0}{r}\right) \\ h_2 &= U\left(1 - \frac{b^2}{aV}\right) \end{aligned} \tag{B.12}$$

B.3. Plotting coordinates into a map

The map used to plot the position is an image[2] containing a scanned map. To plot the position into the map, it is necessary to establish a relation between map points and positions.

[2] It can be a .png, .jpg, .bmp, etc. In this case the image is a .jpg.

Given that the map is an image, it is characterised by its pixels that form a matrix. So, relating pixels to coordinates it is possible to plot positions in the map. Just three control points are necessary for this.

The relation is:

$$\begin{aligned} \text{ph} &= f(\lambda,\phi) = a_0 + a_1\lambda + a_2\phi \\ \text{pv} &= g(\lambda,\phi) = b_0 + b_1\lambda + b_2\phi \end{aligned} \tag{B.13}$$

There are three different matrixes:

- **P**: The control points pixels.
- **C**: The transformation coefficients.
- **A**: The control points coordinates.

Using the control points **A** is calculated, and any coordinates will be able to be calculated as pixels:

$$\underline{\underline{A}} = \underline{\underline{C'}} * \underline{\underline{P}} \text{ where: } \underline{\underline{C'}} = (\underline{\underline{C^T}} * \underline{\underline{C}})^{-1}\underline{\underline{C^T}} \tag{B.14}$$

Knowing A, any pixel is found as:

$$\underline{\underline{P}} = \underline{\underline{C}} * \underline{\underline{A}} \tag{B.15}$$

APPENDIX C. ASCII CODE

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	NUL	32	20		64	40	@	96	60	‘
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	”	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	’	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	TAB	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

Table C.1: ASCII table

BIBLIOGRAPHY

Books

[1] Bleck, N. et all. *Global Positioning System basic concepts. Guide to GPS Positioning*. University of New Brunswick Graphic Services. Fredericton, New Brunswick, Canada. 1987.

Manuals and device documentation

- GPS-Basics-English(GPS-X-02007).pdf
- GPS-Dictionary-Reference(GPS-X-00001).pdf
- Installation-Guide(GPS.G4-CS-05009).pdf
- ANTARIS-SBEKit-Prod-Summary(GPS.G3-PS3-03013).pdf
- ANTARIS-SBEKit-User-Guide(GPS.G3-EK-04013).pdf
- u-center-AE-User-Guide(GPS-SW-02001).pdf
- MHX-920 Operating Manual.pdf
- Netmedia Servo_Torque_Board.pdf

Web pages

- General information: http//www.wikipedia.org
- GPS:
  - http://www.u-blox.com
  - http://www.dbartlett.com/index.htm#intro
  - http://edis.ifas.ufl.edu/IN653
  - http://www.colorado.edu/geography/gcraft/notes/gps/gps_f.html
  - http://spaceflight1.nasa.gov/realdata/sightings/
- NMEA format:
  - http://www.gpsinformation.org/dale/nmea.htm
  - http://www.elgps.com/ftpgps.html#NMEA
- Telemetry: http://www.l-3com.com/TW/tutorial/what_is_telemetry.html
- Datums
  - http://www.ngs.noaa.gov/CORS-Proxy/Glossary/xml/NGS_Glossary.xml
  - http://recursos.gabrielortiz.com/
- Serial port
  - http://www.taltech.com/TALtech_web/resources/intro-sc.html#cablenulls
  - http://pinouts.ru/connector/

  - http://cnx.org/content/m12293/latest/
  - http://www.roborealm.com/help/Serial.php
- Programming:
  - http://www.ensta.fr/∼diam/tcl/online/tcllib/keyword-index.html#KW-nmea
  - http://sunsite.ualberta.ca/Documentation/Gnu/libstdc++-2.90.8/html/21_strings/howto.html
  - http://www.cab.u-szeged.hu/LDP/HOWTO/Serial-HOWTO-15.html#ss15.1
  - http://www.docmirror.net/es/linux/howto/misc/Programacion-Serie-Como/Programacion-Serie-Como-3.html
  - http://www.elotrolado.net/showthread.php?s=&threadid=135524
  - http://www.gp32spain.com/foros/archive/index.php/t-10729.html
  - http://articulos.conclase.net/jm/prog/cpp/cstring.html
  - http://www.cplusplus.com/reference/clibrary/cstring/
  - http://www.flounder.com/cstring.htm
  - http://www.cppreference.com/cppstring/index.html
  - http://www.bgsu.edu/departments/compsci/docs/string.html
  - http://www.yolinux.com/TUTORIALS/LinuxTutorialC++StringClass.html
  - http://www.fredosaurus.com/notes-cpp/
  - http://cimg.sourceforge.net/index.shtml
  - http://www.javielinux.com/sdl_animation/
- Servomotors
  - http://www.seattlerobotics.org/guide/servos.html
  - http://scmstore.com/Servos/controller/index.htm
  - http://robotica.chillan.ubiobio.cl/∼miguel/arcos/index.php?seccion=capitulo4.txt
  - http://www.superdroidrobots.com/product_info/RC.htm#Servos
  - http://robots-argentina.com.ar/Prueba_ServoRC01.htm

Other documents

- González Arbesí, José M. *Proyección de datos sobre un mapa ráster*.
- Martínez Baena, Javier. *Introducción a SDL (Simple Directmedia Layer)*.